ISSN:0975-9646

Model Driven Engineering Using UML. A Pragmatic Approach

Liviu Gabriel Cretu

[#]*Business Information Systems Department,*
Alexandru Ioan Cuza University of Iasi
Bd. Carol I, no 11, Iasi, Romania

Abstract— **In the pursuit of systematic integration of Model Driven Engineering (MDE) principles within the usual software development process, one of the first questions arising is the definition of the MDE process itself. This paper introduces a pragmatic method to apply MDE using UML - the de-facto modelling standard in software engineering. The method presents a well define process based on meta-modelling strategies and UML profiles. MDE shortcuts are also introduced as a mean to facilitate gradual adoption and integration of MDE techniques within the software development process. An example is provided to validate and illustrate this method.**

Keywords— **Model Driven Architecture, Model Driven Engineering, Model Driven Development, meta-models, UML, UML profiles.**

I. Introduction

More than a a decade ago, Object Management Group (OMG) proposed the Model Driven Architecture (MDA™) [1] to deal with the separation of platform dependent and independent aspects in information systems and the transformation rules between them. Since then, MDA has become a well established discipline both in practice and research in information systems and software engineering. However, since MDA is a proprietary trade mark specifying the process and artifacts to be used, people cannot easily modify and adapt to their needs while still using the brand name. A similar situation is, for example, the use of RESTful services and Web APIs where the first imposes strict rules on creating HTTP-based services while the later is just general enough to cover any kind of services accessible via HTTP. As a consequence, today we have multiple acronyms in use for model-driven paradigm:

- MDA – Model Driven Architecture – a process with three translation steps: Computational Independent Model (CIM) to Platform Independent Model (PIM) to Platform Specific Model (PSM) and finally to the generated code;
- MDE – Model Driven Engineering [2] – the usual general term used instead of MDA. It proposes the same approach of modelling multiple abstraction layers and translation rules as MDA, only there is no recommendation regarding the number and content of these layers. MDE is being increasingly promoted as the discipline to manage separation and combination of various kinds of concerns in software or data engineering.
- MDD – Model Driven Development – focuses more on code generation instead of multiple modelling layers. It is usually seen as a two steps process: from model to code.

This paper will present an MDE method specifically designed to seamlessly adopt the principles of model-driven paradigm within the day-to-day software development process using the de-facto standard in software modelling: UML. However obvious it may sound that UML should be used with MDE, the literature mainly offers domain-specific model-driven examples or partial solutions to specific issues. Actually, one cannot easily find a step-by-step guide in MDA/MDE with UML. And this is the main rationale of this paper. We will start with a short literature review, then the MDE with UML method will be presented along with a list of useful MDE shortcuts, and finally an example will validate the proposed method.

II. Model-Driven Engineering Practices

A quick literature review on the subject of MDA/MDE reveals two main areas of research:

1) Domain-Specific Languages (DSL)

2) Meta-models and UML profiles

Most of the domain engineering methodology emphasizes domain modelling as an important mechanism for the development of software systems made of software products (components) with similar architecture. Domain-Specific Languages are specifically tailored to directly represent the concepts of an application domain as programming primitives. Domain-specific languages lift the platform's level, reduce the underlying APIs' surface area, and let knowledgeable end users live in their data without complex software-centric models [8]. We can find DSLs combined with MDA principles used in the development of different types of software. For example, HyperDe is presented in [4] as an environment that supports the design and implementation of web-based applications combining model-based development with domain specific languages for flexible and rapid prototyping of applications. Moreover, in [5] one may find an interesting approach where MDE is applied to compose "programs" written in different DSLs, which will enable the use of the DSL approach to build applications spanning different domains.

The second widely recognized approach is to put meta-models at the very base of the MDA principles [6] and to incorporate them in the software engineering process using manual or automated model-to-model transformers. Meta-models are intended to define a set of related concepts and each meta-model defines a language for describing a specific domain of interest. The associated transformers use this language to generate new models from input models by interpreting the concepts in the meta-model.

Since UML is the standard in software engineerign, the first question is how can one define meta-models with this language, associate them with domain models and apply MDE transformations. In UML, a model element may specify a relationship to the meta-model elements by means of stereotypes and tagged-values. These are modelled using UML profiles. Profiles can play a particularly important role in describing the platform model and the transformation rules between models according to MDE principles. XMI [7] may then be used to transfer meta-models from one project to another, no matter the modelling tool, as long as it is UML based.

Regarding the usage of UML with model-driven paradigm , there are some works showing the natural relationship between UML profiles and the meta-modelling phase in MDA [9, 10] while a large number of papers are proposing domain specific profiles such as for critical infrastructures [11], distributed service models[12], embedded systems [13], web services [14], semantic web services [15] etc. An important number of papers are also dedicated to special languages needed to define and execute MDA Transformers such as in [15].

III. A PRAGMATIC METHOD FOR MDE USING UML

MDE can leverage the software development process only if the latter does follow a set of well-defined principles:

- **Reference architecture** - the system has a clear architecture which both is well documented and a development framework has been built around it. This will provide the meta-model for the new systems. In other words, the development team will never have to raise any questions of the kind: "where should I put this piece of code?"
- **Typed Use Cases** - development tasks are usually organized around Use Cases and if the Use Case is associated with a meta-model (also known as pattern) then we call it a Typed Use Case (TUC). Typed Use Cases lead to typed development tasks for which the domain model abstractions as well as the software pattern to be implemented are known. In these cases the implementation can be estimated with very high level of accuracy, both in time and quality of the work.
- **Just-enough automation** – although MDE does not necessary mean any automation, this is one of the usual goals. In such case the development process automation will not have to generate 100% functional code. The trick is to automate the routine development work and to let the designers and developers concentrate on specific details or some behavioural exotic algorithms. The automation has to focus on two main areas: productivity and bug-free product. For example, generating the main interfaces as well as concrete class structures with inheritance to some abstract behaviour may lead to a productivity boost for junior developers as they will easily add the specific behaviour only in the right place. Not to mention they will also quickly learn the structure of the code for that type of Use Case.

Once we have a reference architecture and TUCs we can apply MDE principles with UML, and even automate the development process, taking into account the pragmatic goal of just-enough automation. UML offers two extension mechanisms very useful for MDE: stereotypes and tagged values. Stereotypes are used to associate UML artifacts with your own meta-model artifacts. Thus, using stereotypes one can further classify Classes, Use Cases, Relationships, and so on in order to bridge the gap between UML meta-model and the target system's meta-model. Tagged values are very useful to specifically define custom association types or other meta-data which have a meaning for an external processor. Both stereotypes and tagged values can be packaged into profiles to create the software development meta-model based on the reference architecture.

The UML-based MDE process is illustrated in figure 1. We still use the MDA artifacts (CIM, PIM, PSM) for convenience. However, since it is an iterative process, there is no restriction on the number of modelling layers.

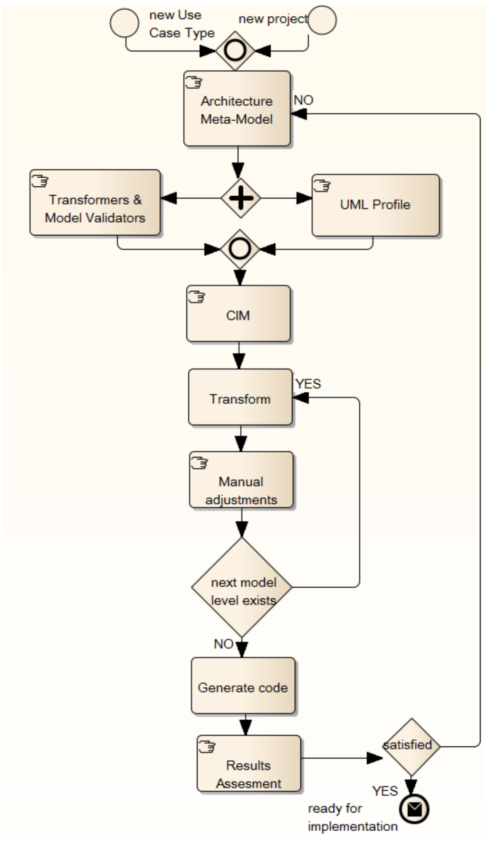

Fig. 1 The MDE process using UML

In short, the UML-based MDE process starts with a meta-model for the system to be developed. This meta-model is derived from the reference architecture of the system. Then, a UML profile (stereotypes and tags) is created together with a set of rules to guide the creation of CIMs and the subsequent transformations. Finally, the stereotypes and tagged values are used to create the individual CIMs (or domain CIMs), models are validated and the transformations are applied (manually or automated).

A. The UML-based MDE process

There are 4 phases of this process:

Phase 1 – create the meta-model based on the reference architecture. The meta-model has three parts: a) an UML profile whose elements will be the labels of the software artifacts to be obtained from each UML element annotated with that stereotype; b) profile usage rules to guide the association of stereotypes and tagged-values to individual CIMs; c) transformation rules to guide the transformation of one model into another. For example, a Use Case may have a kind of relationship to the domain classes in order to specify the input/output parameters. Then, the transformation rules specifically state how the Use Case will be transformed into a Service (PIM), then into a Web Service, RESTful Service or EJB Service (PSM). If MDE automation is the goal, then a collection of Transformers is also created in this phase.

Phase 2 - the business analyst will create the domain analysis CIMs using standard UML elements annotated with the meta-model elements defined in Phase 1. The number and the types of diagrams to be used in this phase will be derived from the reference architecture of the system to be developed. The only two important things to note are the followings: UML elements have to be annotated with meta-model stereotypes and the relationships needed by the MDE process have to be properly defined (based on the same meta-model elements) in order to generate the required modelling or code artifacts. In this phase the developers may also be involved to enrich the models for the MDE Transformers. A set of MDE shortcuts, as defined bellow, may be used.

Phase 3 – transformation rules are applied (manually or automated) to generate PIMs and then PSMs and finally the code. Even if the process is automated, specific manual adjustments may be needed before each transformation. Also, model validators should be defined to check the meta-model semantics associated with the domain CIM.

Phase 4 consists in the analysis of the results, progress assessment, and refinements with the final goal to obtain a higher degree of control and predictability of the development process. Among many tools, CMMI [18] proposes one of the most trusted methods to measure this kind of progress.

B. MDE shortcuts

Code generation implies working with highly formalized models, thus leaving no place for ambiguity. However our experience shows that strict MDA compliance may be quite undesirable in practice. Not only that the distinction between PIM and PSM is vague for most of the developers (mainly because they are using the same technologies and platforms for a long time) but also the time spent to put the Transformers stack in synchronization one with another may simply not be accepted by the management team.

To address these kinds of pragmatic issues, the MDE method proposed in this paper takes into consideration what we have called MDE Shortcuts. An MDE Shortcut may be defined as a systematic usage of links between elements appearing in different models for different viewpoints (e.g. a CIM element may have a link to some PIM element). There are three valid such shortcuts which may be taken into consideration:

1) *A CIM element may have a link to PIM or PSM* elements (even if the later may have been obtained by means of transformations and the link is added afterwards, just before generating the next model). This shortcut is obviously needed when one needs to reuse some existing components or services or add new modules to an existing system. The natural way to go is to reverse engineer the code to PSM. If one will have no time to define the required reverse-transformers till the CIM level, one will surely still need to use the existing classes in order to create the new extension of the system. Another scenario for this shortcut comes from the natural order of steps in software development: having the concept of a Service clarified, the first step will be to implement the Service then the Client (usually the user interface) that will connect to the Service using the provided interface. When one describes the Client's behaviour in CIM, there are two options: a) to link somehow the Client model elements (CIM) to the generated or re-engineered Service interface in PSM, or b) to add tagged-values specifying the concrete interfaces to be used later in the transformation process. In practice, we have found that the first approach seems more appropriate as it provides a unified way for models transformations (first iteration generates the Service models and code, then the second iteration comes back to Client CIM and adds the links to the new Services).

2) *A PIM element may include PSM concepts* – since CIM describes the business logic from the business viewpoint, to be able to generate some code one will have to enrich the model with enough technical information needed by transformers. This process is very much like writing code: no room for ambiguity. As a consequence, the CIM usually needs to incorporate enough information for direct code generation. Moving this information from one model to another may become quite a risky and error prone job. As such, the PSM operation implementation can be generated from the beginning (CIM-to-PIM Transformer) and attached to the corresponding class until the final code generation (usually as a tagged value or a scenario implementation UML element).

3) *Developers may interfere with the MDE multiple transformation* steps in order to add necessary features to PIM/PSM models, before code generation. This way, specific adjustments that have not yet been captured by Transformers will bring the opportunity to obtain 100% executable code.

By using MDE Shortcuts the number of Transformers (and consequently the eventual logical mappings errors) may decrease dramatically, while still keeping enough models to coherently describe the software from all the required perspectives. Following this pragmatic MDE approach proved to bring the promised productivity boost in practice.

IV. MDE WITH UML APPLIED

In order to validate the proposed method we take the example of a business application which needs to implement various business processes. We will name it the Alpha system and we will take a simple example of a business process shown in fig 2. It is an over-simplified order management process where each morning a service gathers all the orders received by means of different channels (e-mail, web site, other systems), the orders then have to be approved by the manager using some web interface and finally the missing items have to be ordered further from the suppliers. To model the Alpha system we use both BPMN and UML since the modelling tool (Enterprise Architect from Sparx Systems) offers a flexible platform and a powerful transformation language to work with both notations. However, UML classical activity diagrams may be also an option with satisfactory results.

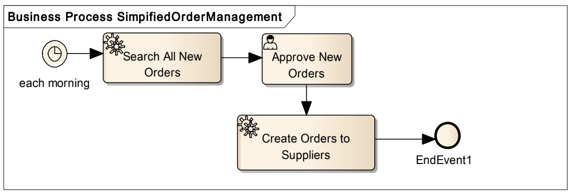
Business Process SimplifiedOrderManagement
each morning
Search All New Orders
Approve New Orders
Create Orders to Suppliers
EndEvent1

Fig. 2 Simplified Order Process using BPMN.

A. The Reference Architecture

According to the process described earlier, we define the reference architecture for the Alpha system (fig 3) as a message based system involving business rules (BR) and business process management (BPM) engines, and an enterprise service bus (ESB).

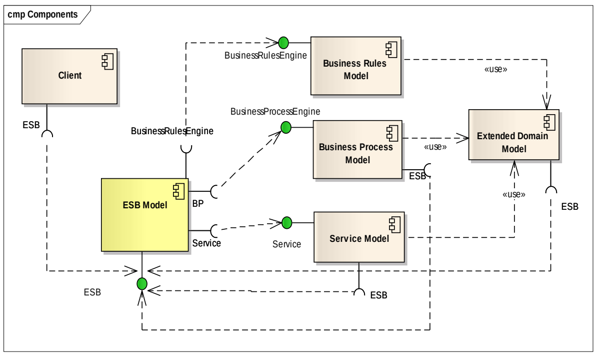

Fig. 3 The reference architecture for the Alfa system

Extended Domain Model – models the internal structure and behaviour of one service. It includes:

1) Domain Model – this is one of most used pattern from Martin Fowler's [17] collection of patterns for enterprise system architectures.

2) Message Model – defines the messages the service may respond to. Each Message corresponds to a business Use Case or Use Case Scenario encapsulating the input data (parameters), necessary for the service execution,.

Service Model – includes those classes that expose the functionality to the world. We call these Domain Services to distinguish them from other services (ESB, business rules, BPM). There is only one public method a Service interface exposes: `handleMessage (message:Message)`. Thus, we call such a service a `MessageHanlder`. Routing one message to the corresponding processing Service will be the ESB's responsibility.

ESB Model – includes components, language and runtime to implement a messaging system, namely to create the configuration of channels, endpoints, routing and transformations to achieve ad-hoc services orchestrations.

Business Process Model – provide components, language and runtime to implement a business process management system.

Business Rules Model – provides components, language and runtime to declaratively define business rules, to associate them as pre-conditions or post-conditions for certain Messages and to execute them against that Message instances when they occur. By separating business rules in a different model, this architecture creates the opportunity for dynamically change the rule set to be applied to one Message instance, depending on the environmental variables accessible from execution context.

Client Model – represents the outside world of the Extended Domain Model. Usually the client refers to the graphical user interface of a system (the presentation layer) or another application/service. Clients execute system's behaviour by sending Messages to the ESB. Thus the client will become dependent only of the Domain Model not the Service Model.

B. The MDE Process

We apply the MDE process defined in the III.A section above for this reference architecture.

Phase 1 - the business analyst will create a CIM version agreed by the customer. Three models will be created in this phase: the Business Process Model, the Domain Model and the Business Rules Model. The Message Model and the Service Model will be generated later on. According to the reference architecture we may create an UML profile as depicted in figure 4. For demonstration purposes, the profile has been simplified to the minimum number of elements needed here. To note: there are two specialized types of <<Service>>UseCases, namely <<Search>> and <<CRUD>> (Create, Read, Update, Delete) with the corresponding message handlers (according to the reference architecture). Dependencies of types Input and Output will be used to specify the input/output parameters for some UseCases (e.g. <<Search>> and <<CRUD>>.

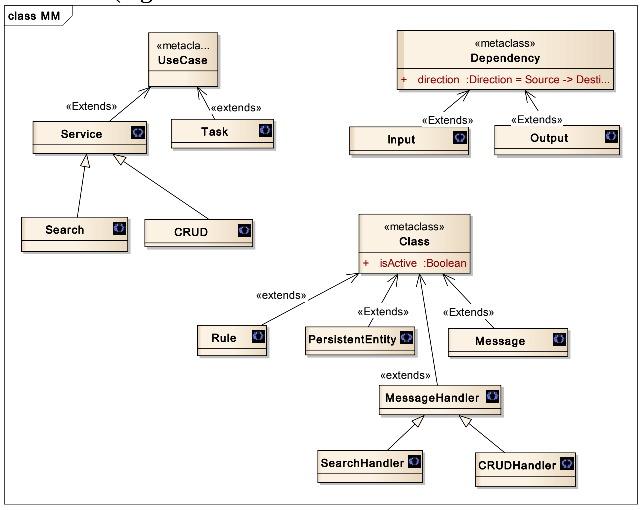

Fig. 4 The UML profile for the reference architecture .

Table 1 shows some of the most important transformation rules we have defined.

TABLE 1.
TRANSFORMATION RULES FOR THE ALPHA SYSTEM

	Transformations	
	CIM	**CIM**
1	BPMN Service Activity	<<Service>> UseCase
2	BPMN Human Task	<<Task>> UseCase
	CIM	**PIM**
	<<Service>> UseCase	<<Message>> Class <<MessageHanlder>>Class implementing the Service interface
	<<Search>>UseCase	<<Message>> Class <<SearchHandler>>Class
	<<CRUD>>UseCase	<<Message>> Class <<CRUDHanlder>>Class
	<<PersistentEntity>>Class	<<PersistentEntity>>Class
	PIM	**<<JavaEE> PSM**
	<<Message>>Class	<<Message>Class
	<<MessageHandler>>Class	<<EJB>>Class implementing the Service interface from reference architecture
	<<PersistentEntity>>Class	<<PersistentEntity>>Class with the Java Persistence API annotations

Phase 2 – the business analyst develops the CIM using the right stereotypes (figure 5). As seen in Table 1, we have one CIM-to-CIM transformation: from BPMN process to UseCase diagram. Once the UseCases are generated, the business analyst may change the <<Service>> stereotype to one of the specialized UseCase types: <<Search>> or <<CRUD>>. This is the case here with the UseCases derived from "Search All New Orders" and "Create Orders to Suppliers" BPMN activities (figure 2). Also in this phase the domain model is created and the links between some UseCases and Classes in order to specify the input and output for some of the UseCases according to the meta-model specification.

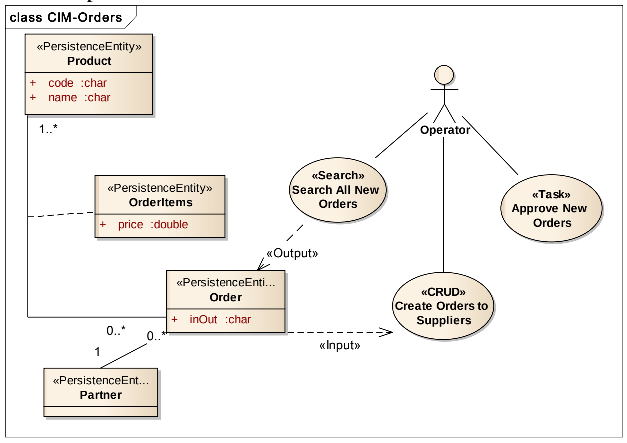

Fig. 5 CIM for Order Process using stereotypes from Alpha meta-model.

Phase 3 consists in executing the transformations and performing the manual adjustments if needed. Based on the rules defined in Table 1, a Transformer may be crated to automate the transformation activity. An example of the result may be found in figure 6. As for the manual adjustments, we apply the MDE shortcuts described earlier. In this example, one manual adjustment was needed: since the <<Input>> and <<Output>> stereotypes are based on Dependency UML meta-class, there is no option to specify the multiplicity of the relationship. Thus, a manual intervention is needed to correct the attribute type of the generated message type. In the specific case of Create Orders to Suppliers, the input may refer to multiple received orders and the output may be a collection of orders to different suppliers.

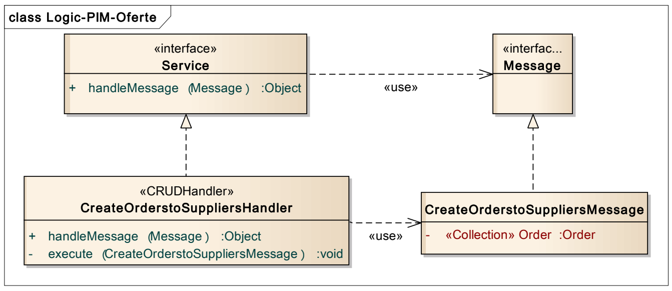
class Logic-PIM-Oferte
«interface»
Service
+ handleMessage (Message) :Object
«use»
«interfac...
Message
«CRUDHandler»
CreateOrderstoSuppliersHandler
+ handleMessage (Message) :Object
- execute (CreateOrderstoSuppliersMessage) :void
«use»
CreateOrderstoSuppliersMessage
- «Collection» Order :Order

Fig. 6 PIM for Order Process generated from CIM (partial).

V. CONCLUSIONS

This paper introduced a well defined method for MDE using UML. The short literature review revealed there is strong orientation, both in research and practice, towards model-driven paradigm. The argument for this work has been the acknowledgement that there is still a lack of such complete guidelines to show how to adopt MDE principles in day-to-day software development business.

We have shown that mastering the relationship between software architecture and UML profiles leads to domain-agnostic MDE process. This is the key aspect which positions this paper as a distinct approach in literature since we have seen a large number of works focusing on domain-specific MDE solutions. The so-called MDE shortcuts has been also presented as valid actions to reduce the number and complexity of MDE Transformers while pushing the level of productivity even further. An example has been provided to better illustrate the process and to validate the method.

REFERENCES

[1] OMG. MDA Guide version 1.0.1. OMG document omg/2003-06-01, 2003

[2] J. Bézivin, "sNets: A First Generation Model Engineering Platform", in: *Lecture Notes in Computer Science*, Berlin, Germany: Springer, 2005 vol. 3844, pp. 169—181.

[3] O. Pastor, S. España, J. I. Panach and Nathalie Aquino, "Model-Driven Development", *Informatik-Spektrum*, Volume 31, Issue 5, pp. 394-407, October 2008.

[4] D. A. Nunes and D. Schwabe, "Rapid prototyping of web applications combining domain specific languages and model driven design," in *Proceedings of the 6th international conference on Web engineering*, 2006, ACM, New York, USA, pp. 153-160.

[5] J. Estublier, G. Vega, A. D. Ionita, "Composing Domain-Specific Languages for Wide-Scope Software Engineering Applications", *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, Berlin, Germany: Springer 2005, vol 3713, pp. 69-83.

[6] J. Bézivin, "In Search of a Basic Principle for Model Driven Engineering", *European Journal for the Informatics Professional,* Vol. V, No. 2, April 2004.

[7] *XML Model Interchange (XMI)*, Object Management Group standard, 1998, http://www.omg.org/docs/ad/98-10-05.pdf

[8] T. Dave, "MDA: Revenge of the Modelers or UML Utopia?", *Software, IEEE, Vol 21*, no. 3 15-17, 2004

[9] F.F. Lidia and A. Vallecillo-Moreno, "An introduction to UML profiles.", in *UML and Model Engineering,* Vol 2, 2004.

[10] O. Rahma and B. Coulette, "Applying Security Patterns for Component Based Applications Using UML Profile." in IEEE 15th International Conference on Computational Science and Engineering (CSE), 2012, IEEE, pp. 186-193.

[11] B. Ebrahim and A. A. Ghorbani, "Towards an MDA-oriented UML profile for critical infrastructure modeling." In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services,* 2006, ACM, p. 66..

[12] S. Raul, F. Fondement, and A. Strohmeier, "Towards an MDA-oriented UML profile for distribution." *In Proceedings of EDOC 2004, Eighth IEEE International Conference on Enterprise Distributed Object Computing,* 2004, pp. 227-239.

[13] S., I. Wisniewski, L. T. Wiedermann Agner, P. C. Stadzisz, and J. M. Simão, "Modeling of embedded software on MDA platform models." *Journal of Computer Science & Technology* Vol 12, 2012.

[14] S., Hassina, I. Bouacha, and M. S. Benselim, "Development of context–aware web services using the MDA approach." In *International Journal of Web Science* vol 1, no. 3, pp. 224-241, 2012

[15] A. B. Djamel and M. Malki, "Development of semantic web services: model driven approach." In *Proceedings of the 8th international conference on New technologies in distributed systems*. ACM, 2008.

[16] M. B. Kuznetsov, “UML model transformation and its application to MDA technology”, *Programming and Computer Software*, Berlin, Germany: Springer 2007, Volume 33, Issue 1, pp 44-53.

[17] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2002.

[18] S. Meena and R. G. Vishwakarma, "CMMI based software metrics to evaluate OOAD." *Proceedings of the Second International Conference on Computational Science*, Engineering and Information Technology. ACM, 2012.